Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability 232.3 (2018): 227 – 247

DOI: https://doi.org/10.1177/1748006X16667328

A Methodology for Model-based Verification of Safety Contracts and Performance Requirements

Elena Gómez-Martínez[†,*], Ricardo J. Rodríguez[‡], Clara Benac Earle[†], Leire Etxeberria Elorza[♯] and Miren Illarramendi Rezabal[♯]

[†]*Babel Group, ETSINF, Universidad Politécnica de Madrid, Spain*

[‡]*Dpto. de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, Zaragoza, Spain*

[♯]*Embedded Systems Research Group, MGEP, Mondragon Unibertsitatea, Arrasate-Mondragón, Spain*

Abstract

The verification of safety requirements becomes crucial in critical systems where human lives depend on their correct functioning. Formal methods have often been advocated as necessary to ensure the reliability of software systems, albeit with a considerable effort. In any case, such an effort is cost-effective when verifying safety-critical systems. Often, safety requirements are expressed using safety contracts, in terms of assumptions and guarantees.

To facilitate the adoption of formal methods in the safety-critical software industry, we propose a methodology based on well-known modelling languages such as UML and OCL. UML is used to model the software system while OCL is used to express the system safety contracts within UML. In the proposed methodology a UML model enriched with OCL constraints is transformed to a Petri net model that enables to formally verify such safety contracts. The methodology is evaluated on an industrial case study. The proposed approach allows an early safety verification to be performed, which increases the confidence of software engineers while designing the system.

Keywords

Safety analysis, Rail system safety, Performance modelling, Modelling/simulation, Life cycle engineering

1. Introduction

The growing adoption of software in safety-critical systems has put safety assessment in the spotlight, becoming a crucial software engineering task as recognised by several initiatives (e.g., the ARTEMIS JU nSafeCer project [1]). Moreover, the design and development of a system must be done with safety in mind, rather than add it in as an afterthought [2]. Several real examples can be found in the literature showing the impact of a lack of safety assessment in industrial systems, such as the flight errors of Air Canada airline, the explosion caused by an overflow in Ariane 5, or the unavailability of the Patriot

* Corresponding author; e-mail: egomez@babel.ls.fi.upm.es

Missile control system in US army base camps [3; 4]. The earlier safety assessment is carried out in a system, the sooner it can be redesigned to properly fulfil safety requirements, thus saving production and other costs.

Contract-based design is a popular approach for the design of complex component-based systems where safety properties are difficult to guarantee [5; 6]. A key benefit of using contracts is that they follow the principle of separation of concerns [7], separating assumptions that the environment of a component obeys from what a component guarantees under such an environment.

The Unified Modelling Language (UML) [8] is widely adopted to model the design of a system. By providing the means to include safety requirements in UML, the integration of safety activities in the normal software lifecycle is facilitated. For safety specification, two approaches have been proposed: (i) to use the Object Constraint Language (OCL) [9], which is a well-known language among the modelling engineering community; and (ii) to use specific UML profiles [10], which extend the semantics of UML models. In [11], the use of both techniques was proposed to express fire prevention requirements of a hospital facility.

Safety requirements modelling is an important part when designing, but some mechanisms must also be provided to assess that safety requirements are indeed fulfilled [2]. To this goal, several formal verification techniques have been proposed (e.g., model checking [6]).

In this paper, we focus on safety requirements and assessment in UML models. Namely, we explore the representation of safety requirements as OCL constraints and their verification using Petri nets [12] as the formal model, obtained after transformation of UML models enriched with OCL, plus UML profiles. By combining standard engineering practice, i.e., UML modelling, with formal verification techniques, i.e., Petri nets, we provide a rigorous safety analysis available for software engineers.

As case study, we evaluate our approach in a real industrial scenario. We model a train door controller with UML, specify its safety requirements, transform these models into Petri nets, and analyse them using well-established analysis tools. The train door controller is in charge of opening and closing train doors and is developed by CAF Power & Automation company[1].

A previous version of this work can be found in [13]. Several enhancements have been carried out with respect to the aforementioned work. In particular, the contribution of this paper is threefold: (i) We propose a three-step methodology for model-based safety verification; (ii) We generalise and formalise the specification of safety requirements as OCL constraints; and (iii) we formalise the transformation from OCL into Petri nets.

Although our aim in this work is to propose a methodology for the full development and analysis of safety and critical systems, there are some current limitations. In this work, the focus is on a set of safety and performance requirements that are represented at a very high abstraction level (components' level). However, consistency between models is not guaranteed. Moreover, some of the tools we used are not compliant with safety standards such as EN5012x and ISO26262, and hence, safety systems cannot be certified by means of our approach. Our approach is notwithstanding useful to safety engineers, since they may assess their designs at early phases, thus reducing development costs.

The rest of the paper contains the following sections. Firstly, Section 2 outlines the basic concepts. Section 3 presents our methodology for model-based verification of safety contracts. Then, Section 4 applies our proposal to an industrial case study. Section 5 discusses obtained results. Finally, Section 6 covers related work and Section 7 states some conclusions and future work.

[1] http://www.cafpower.com/es/

2. Previous Concepts

2.1. UML

The Unified Modeling Language (UML) [8; 14] is a semi-formal general-purpose visual modelling language used for specifying software systems. In this paper, some knowledge of UML is assumed. For more details we refer to [8; 14].

UML can be tailored for specific purposes by profiling. Profiling was introduced by UML to indeed add new capabilities to the language. A UML profile is a UML extension to enrich UML model semantics defined in terms of: *stereotypes* (concepts in the target domain), *tagged values* (attributes of the stereotypes) and *constraints* (formulae that apply to stereotypes and UML elements to extend their semantics). Numerous UML profiles can be found in the literature targeting different specific domains and non-functional properties system analysis (e.g., performance, dependability, security, etc.). For instance, MARTE (Modeling and Analysis of Real-Time and Embedded systems) profile [15] provides support for schedulability and performance analysis in real-time and embedded systems, while DAM (Dependability Analysis and Modelling) profile [16] supports dependability analysis and SecAM (Security Analysis and Modelling) profile [17] focuses on security aspects. In this paper, we use the MARTE profile to indicate the duration of activities in a UML model.

Another extension to enrich UML semantics is the Object Constraint Language (OCL) [9]. OCL is briefly introduced in the following section.

2.2. Object Constraint Language

The Object Constraint Language (OCL) [9] is a formal language used to describe constraints on UML models. The main purpose of OCL is to provide additional relevant information to a UML diagram while avoiding ambiguities arising from the use of informal specification languages. Compared to other formal languages, OCL is sufficiently simple as to be usable in an industrial setting.

An OCL expression can be adopted at four different levels: at classes, at specific class methods, at attributes, or at association roles. An OCL expression provides a textual description about what it is expected or what it is performed by a system. Note that an expression may incorrectly describe the system behaviour by a lack of consistency and coherence between OCL expressions and system's activity.

Unfortunately, a UML model annotated with OCL and a profile that provides support for non-functional properties specification is not a suitable model to quantitatively or qualitatively evaluate these properties. To this aim, we propose the use of Generalized Stochastic Petri Nets, which are introduced in the following.

2.3. Modelling and Analysis of Real-Time and Embedded systems

UML lacks the quantifiable notion of time, so that a UML design lacks the ability to adequately represent time concepts, which is a fundamental feature in the QoS evaluation of a system. To overcome this limitation, the UML-MARTE (UML Profile for Modeling and Analysis of Real-Time and Embedded systems) defined by [15] provides a framework within UML that overcomes this problem introducing such capabilities as annotations (stereotypes and tagged values).

According to UML, each stereotype is made of a set of tags which define its properties. For example, the *gaScenario* stereotype has *respT* as tagged value of MARTE to indicate the response time to be predicted in this scenario. Time durations of activities are specified by means of *gaStep* stereotype and the *hostDemand* tagged value. The values assigned to tags are either basic UML types or NFP types expressed using the Value Specification Language (VSL) syntax. In particular, for complex NFP different values can be set: a value or variable name prefixed by the dollar symbol (*value* property); the origin of the NFP (*source*), e.g., a requirement (*req*), a calculated parameter (*calc*), an estimated (*est*) or measured (*mea*) value; the type of statistical measure (*statQ*), e.g., a mean or a variance. VSL enables the specification of variables and complex expressions according to a well-defined syntax.

2.4. Generalized Stochastic Petri Nets

In this paper, we consider Petri nets [12] as the formal modelling language. More precisely, we translate the annotated UML diagrams into Generalized Stochastic Petri Nets (GSPNs) [18], following the guidelines proposed in [19].

A GSPN is a graphical and mathematical formalism used for the modelling of concurrent and distributed systems. Informally, a GSPN is a bipartite graph of places and transitions joined by arcs, describing the flow of the system with concurrency and synchronous capabilities. Graphically, places and transitions are respectively represented by circles and bars; while arcs are depicted by directed arrows. Places can hold tokens (graphically represented by a black dots or by a number inside a place) that represent system resources or system workload, while transitions represent system activities. A transition is enabled when its input places hold enough tokens. The firing of enabled transitions represents a change in the system state. When a transition fires, tokens from input places are removed and placed in output places. The weight of arcs connecting these places and transition determine how many tokens are removed/placed. A GSPN distinguishes two kind of transitions: immediate transitions, which fire at zero time (i.e., its firing does not consume any time); and timed transitions, which may follow different firing distributions such as uniform, deterministic or exponential distributions. In this paper, we consider timed transitions with exponentially distributed random firings. Immediate transitions, depicted as thin black bars, can have also associated probabilities to represent the system routing alternatives. Exponential transitions, drawn as white boxes, account for the time that takes an activity to complete. Furthermore, there exist different semantics for the firing of transitions; infinite and finite server semantics being the most frequently used. Since infinite server semantics is more general (finite server semantics can be simulated by adding self-loop places), we will assume that the exponential transitions work under infinite server semantics.

A GSPN defines also a function $\Pi : T \rightarrow \mathbb{N}$ that maps transitions of the Petri net system onto natural numbers representing the priority level of each transition. The priority of a transition t is indicated by $\pi(t)$. In case of transitions enabled at the same marking, the transition with highest priority fires first. Similarly, a GSPN also allows to define marking-dependent enabling functions for transitions. These functions return a value of 1 to indicate whether the transition is enabled, 0 otherwise. Enabling conditions of a transition t are depicted as boolean formulae between square brackets. In these functions, the marking of a place p is denoted by $\#(p)$.

3. A Methodology for Model-based Safety and Performance Assessment

In this paper, we present a scenario-based methodology to verify performance and safety requirements at early stages. The proposed methodology is an extension of the performance analysis methodology presented in [20]. The performance assessment methodology has been evaluated in several case studies [21; 22; 20]. Both performance and safety methodologies follow Software Performance Engineering (SPE) principles and techniques [23].

The proposed joint methodology uses UML diagrams [8] enriched with performance information by means of the MARTE profile [15], and safety constraints expressed in OCL [9]. For the assessment, Petri nets are used (namely, GSPN [18]). Therefore, this merged methodology allows safety and performance assessment at early design phases. This assessment is in fact obtained as a "by-product" of the software life-cycle.

The proposed methodology depicted in Figure 1 comprises five different phases, which are outlined in the following paragraphs:

1. **Design.** The software system is modelled using UML diagrams, mainly UML Composite Diagram (UML-CoD), UML Class Diagrams (UML-CD), UML Interaction Overview Diagrams (UML-IOD), UML Sequence Diagrams (UML-SD), and UML State Machine Diagrams (UML-SM). This phase captures the main system structural elements, their behaviour, and interactions. As results, a set of UML diagrams representing the software system is obtained.

2. **Specification**. This phase captures non-functional requirements of the critical system. It is composed of two parallel activities:
   - **Performance Specification.** In this phase, UML diagrams are annotated with performance information according to the MARTE profile. Scenarios which may become crucial for the performance of the system are identified.
   - **Safety Specification.** Each scenario where safety issues may arise is further specified by means of safety requirements. Safety requirements are first expressed as Safety Contracts Fragments [24], and then translated into OCL within UML models.

   As results UML diagrams annotated with OCL and MARTE are obtained in this phase.

3. **Transformation into a Formal Model.** In this phase, UML models enriched with MARTE annotations and OCL constraints are translated into a so-call *performance and safety model*. Specifically, we use the formalism of GSPN [18]. As result, we obtain a GSPN that represents a safety-critical system, accounting for performance and safety issues.

4. **Analysis.** The aforementioned GSPNs are analysed to obtain measures of interest, such as response time, scalability, or occurrence probability of undesired states. These measures can be obtained by analysis or simulation of the model. We use the GreatSPN [25] tool for computing them.

5. **Assessment.** Lastly, the assessment phase verifies whether performance and safety requirements are fulfilled, which helps system designers to consider alternatives for improving system design from a performance-safety perspective.

Note that this methodology for assessing safety-critical systems is transparent to software designers, that is, a software designer should not be concerned with the formalism and tools used in the analysis phase, but rather focus on the design and specification phases.

In the sequel, we describe in detail these phases. The proposed methodology is evaluated in Section 4, where safety properties of an industrial case study are verified.

3.1. Design Phase

The first step in our methodology is to model a software system, as well as its behaviour. Since reuse of software components is key in, for instance, aerospace and automotive domains [26], a component-based design is followed. We use UML [8] as the modelling language for software design. We mainly focus on those UML diagrams which allow us to extract performance or safety information for the next steps of our approach: Class Diagram (UML-CD), Use Cases (UML-UC), Deployment Diagram (UML-DD), Interaction Overview Diagram (UML-IOD), Sequence Diagram (UML-SD), Activity Diagram (UML-AD) and State Machine Diagram (UML-SM). For the component-based design, we also use Composite Structure Diagrams, in particular, the SysML [27] extension. The reason is that in SysML input and output ports can be defined and they are used in the safety contract as explained in Sect. 3.2. The following paragraphs explain how we use each of these diagrams. For a more extended explanation of these UML diagrams, see [8].

- **UML Class Diagram** (UML-CD) describes the static structure of a system in terms of classes and relationships between classes. Classes are essentially organized through aggregation, inheritance or association relationships.
- **UML Composite Structure Diagram** is a type of static structure diagram which represents the internal structure of a structured classifier or collaboration to describe a functionality. Thus, a Composite Structure Diagram represents runtime instances collaborating over communication links to achieve some common objectives. This diagram can include, among others, *parts*, a set of one or more instances which are owned by a containing classifier instance, and *ports*, which define a property of a classifier that specifies a distinct interaction point between that classifier and its environment or between the (behaviour of the) classifier and its internal parts. Ports may specify inputs, outputs

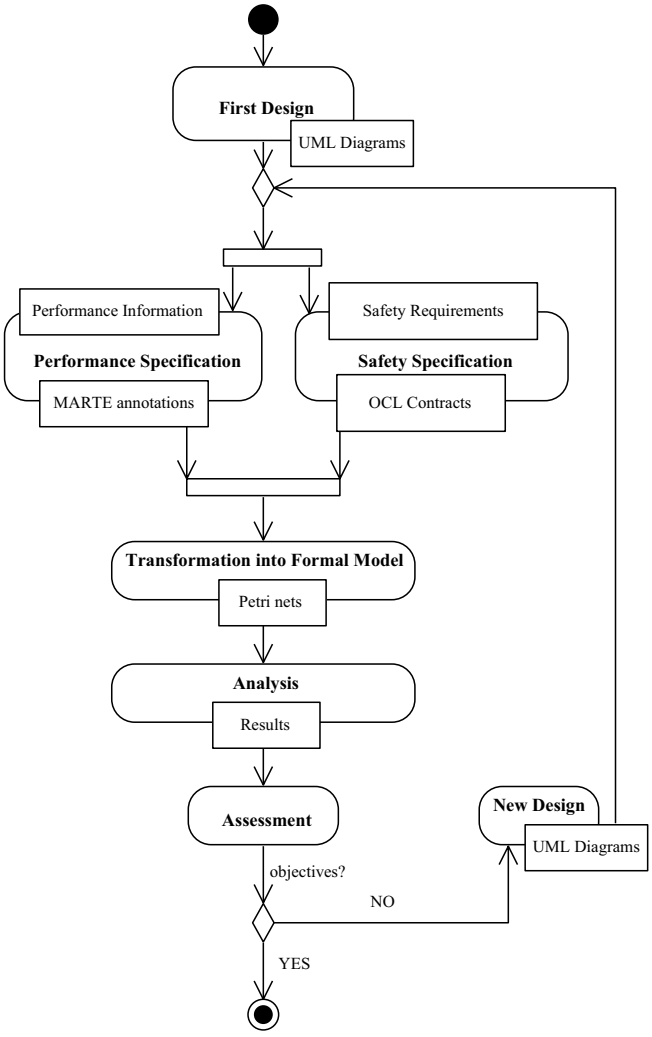


Fig. 1. Methodology for model-based safety analysis.

as well as operating bidirectionally. In contract-based design each safety critical component of the system and non-critical components are seen as separated components [28] which interact with their environment. In the proposed methodology, we use Composite Structure Diagrams to represent components and subcomponents of a critical system.

- **UML Use Case Diagram** (UML-UC) is a behavioural diagram that describes the functionality of a system in such a way that shows all available functionalities. In our approach, UCs model *usage scenarios*, as well as the population, that is, the number of concurrent users in each scenario if this would be the case.
- **UML Interaction Overview Diagram** (UML-IOD) is a special and restricted kind of UML Activity Diagram which represents the flow relationships among fragments and UML Sequence Diagrams. IODs represent high-level scenarios, as well as their population.
- **UML Deployment Diagram** (UML-DD) identifies the system software components as well as the hardware nodes in which the former are deployed. DDs are used to have a static view of the software architecture and, for the performance perspective, to show potential communication delays and/or the available resources.
- **UML Sequence Diagram** (UML-SD) shows object interactions; more specifically the messages exchanged between the system components arranged in time sequence. It provides useful constructors such as loops, alternatives or parallel execution. It is used to model usage scenarios with respect to a timeline. SDs represent the behaviour of usage scenarios with time information, host demands, messages size exchanged and acquisition/release of resources.
- **UML Activity Diagram** (UML-AD) specifies the control flow of a component, subsystem, or system. An activity represents an action in the execution of the activity. In our methodology, ADs are employed to express execution times of actions within a specific activity, as well as the acquisition/release of concrete resources.
- **UML State Machine Diagram** (UML-SM) describes the lifetime of objects. A state represents a time period in the life of an object during which the object satisfies some condition, performs some action or waits for an event. SMs are mainly used to get information concerning activities duration. In component-based design, the internal states of a component are modelled using a UML-SM.

The software engineer decides which of these UML diagrams better express the performance and safety design of the software system, as well as the level of detail, i.e., an IOD shows high-level interactions while a SM specifies object level. Nevertheless, from the performance and safety perspective, it is sufficient to model a UML-CS and at least one behavioural view (IOD, SD, AD and/or SM).

3.2. Specification Phase

Once the system is designed and modelled by means of UML diagrams and, following SPE principles, we now introduce the performance and safety specification of the software system, i.e., information that is relevant from performance and/or safety perspective. We focus on potential critical scenarios, which must meet both safety and performance requirements. Recall that this phase is divided into two parallel activities (or steps), which are further described in the following paragraphs.

Performance Specification Following SPE principles, we introduce performance specification by annotating design diagrams. Performance information allows performance objectives to be predicted. Performance objectives not only include performance metrics with a specific threshold; it is typically a set of numbers describing the context for a particular situation in a scenario from a performance perspective, called performance scenario. Performance objectives can be expressed in several different ways, including response time, throughput, or constrains on resource usage [23]. We specify performance information by means of the MARTE profile [15] and, specifically, the Generic Quantitative Analysis Model (GQAM) framework. MARTE annotations capture properties, measures and requirements of interest for carrying out performance analysis.

We use a subset of MARTE annotations detailed in [29]. In our proposal, MARTE annotations account for:

- The performance measure to be computed, which is the response time.
- The system's closed workload, which describes the number of users that will concurrently populate the system.
- The host demand an activity consumes.
- The routing rates in the system which are expressed as probabilities.
- The utilization of the shared resources, that respectively describe the acquisition and release of the resources.

Safety Specification The second step encompasses the definition of safety requirements to be verified. In this paper, we assume that safety requirements are informally captured from natural language and formally specified as Safety Contract Fragments (SCF) [24]. These SCF are later transformed into OCL and integrated into UML models to be analysed in the next phase.

A SCF defines a safety contract as a set of assumptions and a set of guarantees, for a given component and under a given environment. An assumption is what is expected to be met by the environment, while a guarantee specifies how the component behaves in such an environment. A component of a component-based system, considering only its interaction with the environment, can be formally defined as:

Definition 1. *A component $\mathcal{C} = \langle \mathcal{I}, \mathcal{O} \rangle$ of a system is composed of a set $\mathcal{I}$ of input ports and a set $\mathcal{O}$ of output ports.*

Given Definition (1), a SCF $\mathcal{S}_\mathcal{C}$ of a component $\mathcal{C} = \langle \mathcal{I}, \mathcal{O} \rangle$ can be defined as:

Definition 2. *A SFC $\mathcal{S}_\mathcal{C} = \langle \mathcal{A}, \mathcal{G} \rangle$ of a component $\mathcal{C} = \langle \mathcal{I}, \mathcal{O} \rangle$ is a tuple of safety assumptions and safety guarantees[2]. A safety assumption $a \in \mathcal{A}$ is a (atomic or composite) logic proposition that relates one or more of the input ports of a component. Similarly, a safety guarantee $g \in \mathcal{G}$ is a logic proposition that relates one or more of the output ports of a component.*

Thus, a SCF $\mathcal{S}_\mathcal{C}$ defined over a component $\mathcal{C} = \langle \mathcal{I}, \mathcal{O} \rangle$ relates the input and output ports of the component with the assumptions (i.e., what it is expected) and guarantees (i.e., what it is performed), respectively. Note that for us how the guarantees are achieved is a black-box operation. Besides, the guarantees are only assured when the assumptions are fulfilled. Otherwise, the result of the component is not guaranteed and thus, cannot be trusted as a well-performed operation.

Recall that OCL is a UML extension to express constraints acting over a context into UML models (see Section 2.2). Among other constraints, an OCL can define invariants (`inv`) as state conditions always fulfilled, or pre/post-conditions fulfilled before/after an operation is performed. In this paper, we focus on OCL invariants.

Definition 3. *An OCL constraint $\mathcal{R} = \langle \mathcal{X}, \mathcal{V} \rangle$ is a tuple conformed by a context $\mathcal{X}$, that represents the context in which is defined, and the invariant formula $\mathcal{V} = \langle ls, rs \rangle$, where ls, rs are two logical propositions interpreted as ls `implies` rs.*

Following the above definitions, we can straightforwardly map an SCF into an OCL constraint:

Definition 4. *An OCL constraint $\mathcal{R} = \langle \mathcal{C}, \mathcal{S}_\mathcal{C} \rangle$ describes a context defined by a component $\mathcal{C}$, and an invariant formula defined by the SCF $\mathcal{S}_\mathcal{C} = \langle \mathcal{A}, \mathcal{G} \rangle$.*

Thus, a SCF defined over a component can be mapped into an OCL constraint, and integrated within UML models. In the next phase, these OCL constraints are transformed into Petri nets to drag the safety requirements into the analysis step.

3.3. Transformation into Formal Model

The next phase of our methodology encompasses the transformation of performance and safety scenarios into GSPNs. This phase in turn is divided into two steps, depending on the model to be transformed: Firstly, UML diagrams with MARTE annotations; and then, OCL constraints.

[2] As in [11], for the sake of simplicity we restrict the logic of SCF assumptions and guarantees to AND and OR logic operators.

Performance Transformation. In this step, we need to obtain a performance model for each critical scenario that we have previously annotated.

As above mentioned, performance models are formal models that help to obtain measures of interest (e.g., system response time) by analysis or simulation. There are different kinds of performance formalisms widely accepted in SPE: queuing networks [30], stochastic process algebras [31] and stochastic Petri nets [18]. There exist SPE methodologies that translate performance-annotated UML models into the aforementioned formalisms. For example, the work in [32] to obtain queuing networks, the work in [33] to obtain process algebras or [34; 35] to obtain Petri nets. Some of these methodologies have associated tools that automate the translation process.

We propose to use stochastic Petri nets (SPN) and concretely generalized (GSPN). Section 2.4 gives a brief introduction of Petri nets formalism. In the following, we assume that the reader is familiar with this formalism. This choice has been driven by two main factors: (i) GSPNs provide a formal notation which avoids any source of ambiguity while representing the stochastic behaviour of systems; (ii) GSPNs have a clear graphical notation and several tools have been developed for analysis (for instance, GreatSPN [25], TimeNET [36], or PeabraiN [37], among others). Moreover, the transformation from UML to GSPN can be carried out using well-established tools, such as ArgoSPE [38], ArgoPN [39] and ArgoPerformance [35].

We translated UML diagrams with performance information augmented MARTE profile into GSPNs using the ArgoSPE plugin developed in [38]. This tool implements the algorithms proposed in [40] and [19]. The entire translation process is detailed in [29]. Summarizing this process, object states and resources are mapped into places within the Petri net. Events and actions are translated into immediate and timed transitions, respectively. The average execution of an action is specified with an exponentially distributed random variable. Alternative fragments or conditions are translated into transition with probabilities.

Safety Transformation. As second step, OCL constraints are transformed into GSPN as follows. This transformation is manually carried out since ArgoSPE does not support this translation yet. Recall that each OCL invariant $\mathcal{S_C}$ of a constraint $\langle \mathcal{C}, \mathcal{S_C} \rangle$ is a proposition of a set of assumptions $\mathcal{A}$ implying (`implies` binary operator, $\rightarrow$) a set of guarantees. The satisfaction of a safety constraint (i.e., the invariant) is transformed to indicate the violation of that invariant by reaching a marking. Thus, assuming $\mathcal{S_C} : \mathcal{A} \rightarrow \mathcal{G}$ we build a representative GSPN model to check whether $\mathcal{A} \wedge \neg\mathcal{G}$ is fulfilled in a single-execution of the Petri net.

The transformation is performed following a top-down approach. First, assumptions and guarantees are processed: for each assumption $a \in \mathcal{A}$ and guarantee $g \in \mathcal{G}$, places p_a, p_g, are created. Then, a place $p_{\neg\mathcal{S_C}}$ representing the violation of $\mathcal{S_C}$ and an input transition $t_{\neg\mathcal{S_C}}$ are added, having a priority π' greater than the priority of any other transition in the Petri net, i.e., $\pi' > \pi(t), \forall t \in T$, where $\pi(t)$ is the priority of transition t. We define the following marking-dependent enabling function f for transition $t_{\neg\mathcal{S_C}}$:

$$f_{t_{\neg\mathcal{S_C}}} = \begin{cases} 1, & f_{\mathcal{A}} \wedge f_{\neg\mathcal{G}} \\ 0, & otherwise \end{cases} \tag{1}$$

where $f_{\mathcal{A}} = \bigwedge_{i=1}^{|\mathcal{A}|} \#(p_a) \geq 0$ ($f_{\neg\mathcal{G}} = \bigwedge_{i=1}^{|\mathcal{G}|} \#(p_g) = 0$) is a boolean formula in conjunctive normal form that verifies whether the marking of $p_a(p_g)$ is greater than or equal to 0, i.e., $\#(p_a) \geq 0, \#(p_g) = 0, \forall a \in \mathcal{A}, g \in \mathcal{G}$.

Note that the inequality of each p_a becomes an equality (i.e., $\#(p_a) = 0$) only when the negation of an assumption a is needed, as it happens with the guarantees, otherwise, $\#(p_a) > 0$. That is, we consider the fulfillment of an assumption/guarantee as having a token in the place that represents such an assumption/guarantee). Similarly, the fulfillment of the negation of an assumption/guarantee means to have no token in such a place.

Currently, the translation of OCL constraints is not automatically integrated in ArgoSPE. Thus, some manual tuning is needed. Once the Petri nets that represent the system and the safety constraints are obtained, they are merged using transitions that set tokens in places that represent the same issue in the Petri net of the system, regardless whether they are expressing the negation of clauses or not: the negation of a clause only indicates the symbol when checking the marking

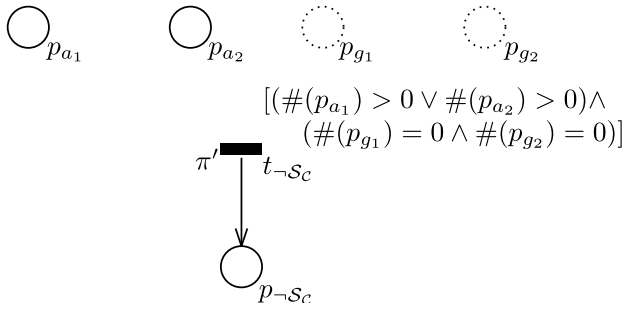


Fig. 2. Transformation from an OCL invariant $\mathcal{S}_\mathcal{C} = \langle a_1 \vee a_2, g_1 \vee g_2 \rangle$ into Petri nets.

of the place. The composition is carried out using the tool called algebra of GreatSPN [25]. As a final step, we manually tune the merged network to ensure that places p_a, p_g and places that represent the same issue are identical, i.e., they have exactly the same input and output transitions and the same initial marking. Recall that a place p is identical of a place $q \neq p$ iff $\mathbf{m_0}(p) = \mathbf{m_0}(q)$, ${}^\bullet p = {}^\bullet q$, and $p^\bullet = q^\bullet$.

The full process of transformation performed guarantees the following:

- The priority of $t_{\neg \mathcal{SC}}$ equal to the maximum priority of the Petri net ensures that, once enabled, this immediate transition is fired first.
- Places $p_a, p_{\neg g}, \forall a \in \mathcal{A}, \forall g \in \mathcal{G}$ being identical to places that represent the same issue ensure that those additional places truly represent when the issue is occurring, at any time.
- The marking-dependent enabling function $f_{t_{\neg \mathcal{S}_\mathcal{C}}}$ ensures that all conditions are evaluated at the same time.

Let us illustrate this transformation with a small example. Consider a $\mathcal{S}_\mathcal{C} = \langle a_1 \vee a_2, g_1 \vee g_2 \rangle$. Four places $(p_{a_1}, p_{a_2}, p_{g_1}, p_{g_2})$ are added and set as identical places to places that represent the same issue in the original Petri net. Then, a place $p_{\neg \mathcal{S}_\mathcal{C}}$ and a transition $t_{\neg \mathcal{S}_\mathcal{C}}$ are added. Let us assume that the maximum priority of any transition of the Petri net is equal to one. Thus, priority π' of $t_{\neg \mathcal{S}_\mathcal{C}}$ is set to 2, i.e., $\pi' = 2$. Finally, the marking-dependent enabling function $f_{t_{\neg \mathcal{S}_\mathcal{C}}}$ is defined as $f_{t_{\neg \mathcal{S}_\mathcal{C}}} = \{1 : (\#(p_{a_1} > 0 \vee \#(p_{a_2}) > 0) \wedge (\#(p_{g_1} = 0 \wedge \#(p_{g_2} = 0)); 0 : otherwise\}$. Figure 2 shows the Petri net generated with the transformation in this example. Places that refers to the negation of an assumption or guarantee are drawn with a dotted line.

3.4. Performance and Safety Analysis

According to the proposed methodology, we carry out safety and performance analysis under steady state assumption in order to validate if the critical system meets safety and performance requirements, respectively. To this aim, we use the GreatSPN tool [25]. Regarding performance objectives, we compute response time by computing the transition throughput, since response time is its inverse. Concerning safety requirements, in terms of Petri nets, we verify whether places that represent violation of safety conditions are eventually marked (i.e., a safety condition is eventually violated) in an execution of the net.

3.5. Assessment

When the safety-critical system does not meet performance objectives or safety requirements, an assessment phase is necessary to re-design and improve the initial design. From the performance perspective, we follow the recommendations detailed in [29; 20]. This systematic performance assessment is based on SPE techniques and techniques [41] and proposes design alternatives, such as resource replication, the application of performance patterns [42], and performance anti-patterns [43]. Regarding safety assessment, we are exploring ways to systematically propose new redesigns. For the moment, this redesign must be done manually by safety experts.

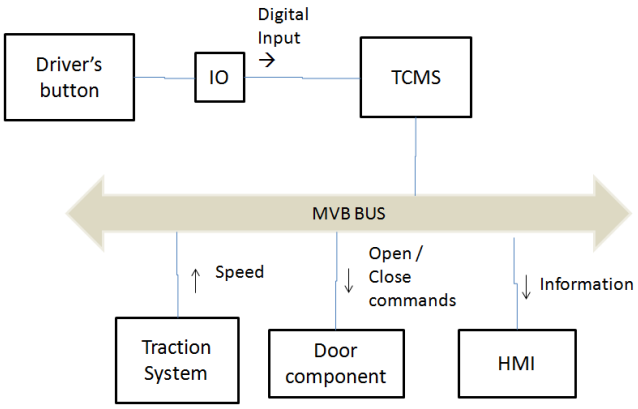


Fig. 3. The TCMS System and others components.

4. Case Study: A Train Doors Controller

In this section, we pursue to perform an early safety verification in a real industrial case study, following the methodology proposed in Section 3.

4.1. System Description

As case study, we consider the door control management performed by a Train Control and Monitoring System (TCMS). The TCMS is a complex distributed system that controls many subsystems such as the door control, traction system control, air conditioning control, video surveillance, passenger information system, etc. The TMCS provides information to the driver, such as the state of doors, the state of the traction, or the state of the alarm system, which is gathered by a set of Input/Output (IO) modules. Figure 3 shows the communication architecture among TMCS and other train subsystems.

The system level requirements concerning the operation of opening and closing of doors are satisfied by the following components:

- The `TCMS` component decides whether to enable or disable the doors considering the driver's requests and the train movement. Thus, doors must be enabled before they can be opened, and disabled before closing;
- The `Door` component controls and commands the opening and closing of a door;
- The `Traction` component controls and commands the train movement; and
- The `MVB` (Multifunction Vehicle Bus) component intercommunicates the components.

Door control systems differ depending on the type of train where they are acting. For instance, a door of a suburban or underground train has a button that enables passengers to open it upon request, while in the case of long distance and high speed trains, doors have no buttons since they are opened only upon the driver's request. In this paper, we consider a door control system in a suburban train, i.e., a door has open buttons inside and outside the train coach. Note that in this case the driver must first enable doors before they can be opened upon passenger's request. Doors also include an obstacle sensor to prevent a closing operation when an obstacle is detected. Figure 4 depicts the door considered in this system.

The train subsystems such as the door control system are safety-critical systems and, therefore, railway standards must be applied during their development. The major standards are the European EN5012x family of railway standards. The following are the three standards relevant to the case study:

- EN50126 [44]: Railway specifications — The specification and demonstration of Reliability, Availability, Maintainability and Safety (RAMS);

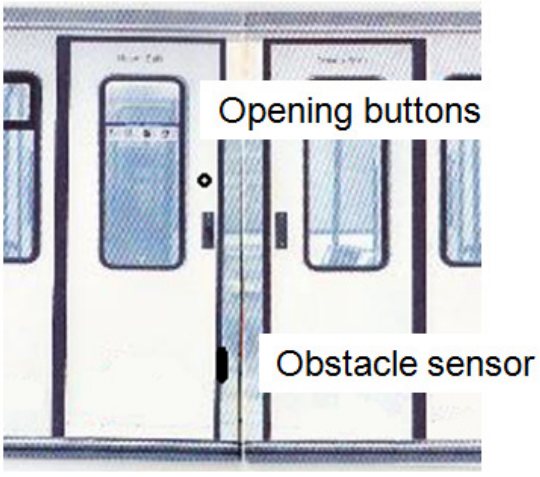


Fig. 4. The image of the door.

- EN50128 [45]: Railway applications — Communication, signalling and processing systems — Software for railway control and protection systems, it is known as the Railway Software Standard and is a specialisation of IEC 61508 for railway;
- EN50129 [46]: Railway applications — Communication, signalling and processing systems — Safety related electronic systems for signalling. EN 50129 gives precise guidance how to build a safety case and particularly what has to be included in the various parts of it;

The Safety Integrity Level (SIL) of the door control system is **SIL 2**. A SIL specifies a target level of risk reduction and is typically defined in components that operate in a safety-critical system [47] [45]. There are four discrete integrity levels associated with SIL with SIL 4 the most dependable and SIL 1 the least. The SIL can be assigned to any safety relevant function or system or sub-system or component. The SIL level allocation is made taking into account the rate of dangerous failures and tolerable hazard rate of the function, system, sub-system or component.

The SIL of a system to be developed is determined on system level (EN 50126). The software "inherits" the SIL as any other part of the system through decomposition. Then, the EN 50128 standard defines what must be done to develop SW functions with that SIL.

In railway domain EN 50129 standard, which explicitly claims to be "the sector specific interpretation of IEC 61508", a safety case is required. A Safety Case is "a structured argument, supported by a body of evidence that provides a compelling, comprehensible and valid case that a system is safe for a given application in a given environment" [48]. A Safety Case should demonstrate the fulfilment of the allocated Safety Integrity Level, SIL2 in this case.

In a compositional approach [1] [49], a safety case would contain the top-level claim about the safety of the overall system, the decomposition into more detailed claims about its constituent subsystems or components, the arguments that show that the components fulfil the safety-related claims that are made about its properties and parts of the component specification may be expressed as component contracts, i. e. as assumptions on the component's environment and guarantees of properties that the component will satisfy when those assumptions are fulfilled. Some of these contracts will address safety-related properties and are referred to as "safety contracts". The safety case will then show, either explicitly or by referencing other documentation that both the assumptions and the guarantees of these safety contracts are fulfilled.

This paper focuses in the safety contracts that will be part of the safety case. A safety contract approach ensures that there is traceability between the evidence provided in the safety argument, the software system design and the system hazards which must be controlled. Establishing such traceability is one of the key challenges for demonstrating the safety of systems [49].

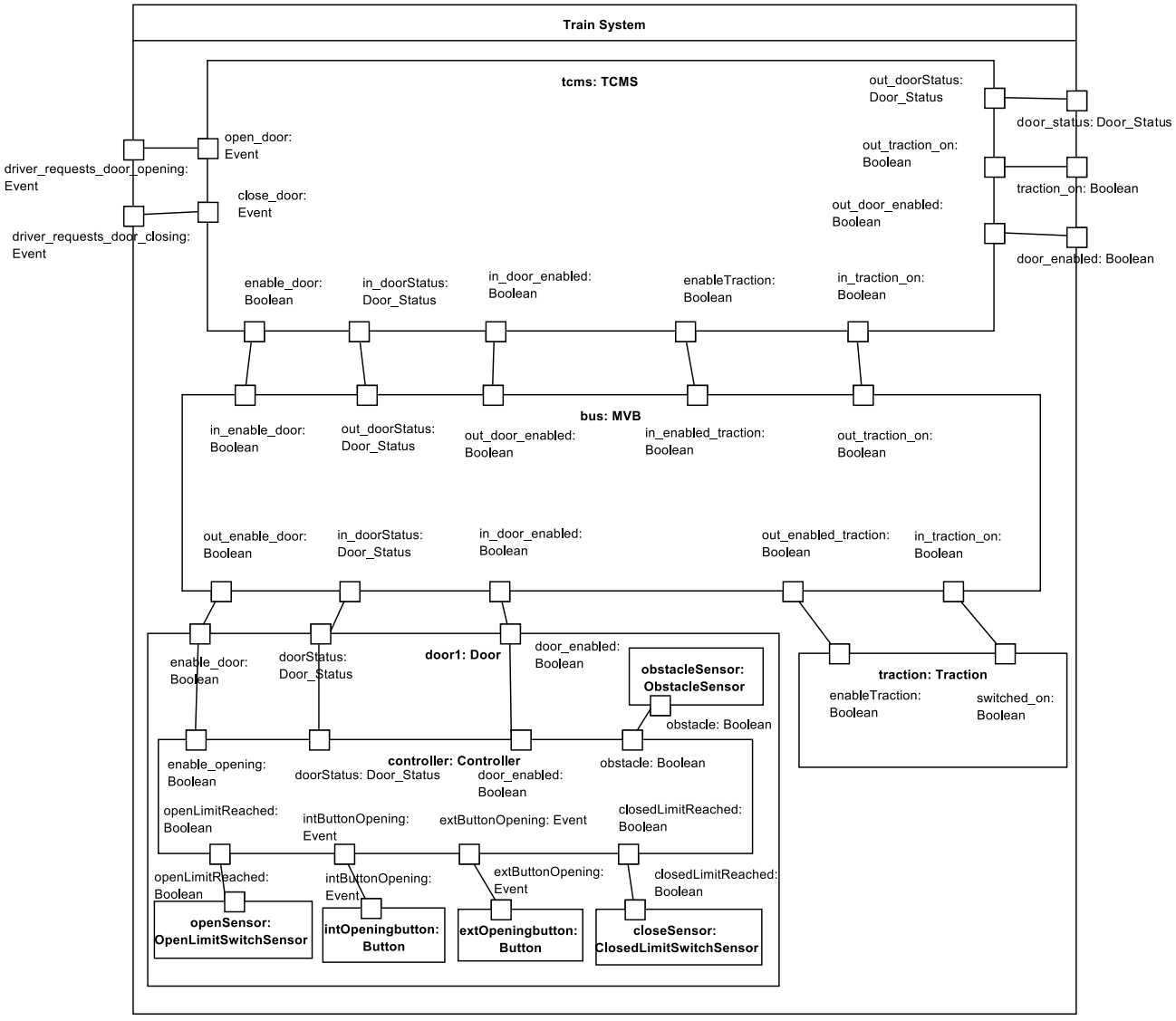


Fig. 5. UML Composite Structure Diagram of the Train System.

The case study presented here concerns a real system where some simplifications were made. Namely, the interaction with other components of the TCMS, the dependencies with other subcomponents, and their communication were omitted. And the main focus is on normal behaviour (although erroneous modes are also described).

4.2. On Design Phase

According to the methodology proposed in Section 3, the first step is to design the critical system. In the following, we describe each safety-critical component in detail.

Figure 5 shows the UML-CS of the Train System. The system is composed by a `TCMS` component, N `Door` components, a `Traction` component and a `MVB` component. The Train System has two external input ports, connected to the input ports of `TCMS` component (namely, *open_door* and *close_door*), which receive the driver requests for enabling or disabling the doors. Output ports of the system report about the status of the overall system: A *door_status* enumerated value to indicate whether the doors are being opened, closed, already open, already closed or error state; a *door_enabled* boolean

value to indicate whether the doors are enabled or disabled; and a *traction_on* boolean value to express whether the traction system is on or off.

As input, the `Door` component receives the command to enable or disable the door (*enable_door* boolean value) from `TCMS` component. As output, the `Door` component reports about the status of the door (*doorStatus* enumerated value), and whether the door is enabled (*door_enabled* boolean value). Note that these outputs are in fact inputs port for `TCMS` component.

`Traction` component receives as input an enable/disable traction command (*enableTraction* boolean value) from `TCMS` component; and provides as output a boolean flag to indicate the traction status (*switched_on* boolean value).

Finally, the `MVB` component represents the communication among the components of the Train System.

Figure 5 also shows the subcomponents of the `Door` component, i.e., the controller (in the following we name it as `DoorController`), the limit sensors, the obstacle sensor, and the interior/exterior opening buttons.

Figure 6 shows the UML-SM of `DoorController`. In its normal behaviour (detailed in the parallel region called *normal*) has four states: *opening*, *isOpen*, *closing*, and *isClosed* (initial state). The interior/exterior opening buttons trigger when pushed the *intButtonOpening* and *extButtonOpening* events, which lead the `DoorController` state to *opening* state, if *enableDoor* is true. Once the door is totally open, `openSensor` triggers an *openLimitReached* event that causes the `DoorComponent` to change to *isOpen* state. It remains in this state until the door is disabled, moving to *closing* state. In this state, two exits are possible: When an obstacle is detected, or the interior/exterior opening buttons are pushed and the door is enabled, the `DoorController` state moves to *opening* state again; When the `closeSensor` component triggers a *closedLimitReached* event, since the door has been totally closed, the `DoorController` is lead to *isClosed* state.

Apart from the normal behaviour of control of the door, erroneous behaviours are also modelled in parallel regions.

The door could not be closed if there is an obstacle in the doorway; if the *obstacle* is there for a long time (*EnduringObstacleTime*) the driver should be warned. The enduring obstacle can be caused by a person or bulk but also it can be due to a problem or failure of the obstacle sensor. This behaviour is detailed in the parallel region called *ObstacleMonitoring* of the state machine.

For detecting errors in the limit sensors (*openLimitReached* or *closedLimitReached*), impossible combinations of sensors states are checked: If the door is open (state *isOpen*) the *closedLimitReached* could not be 1, if the door is closed (state *isClosed*) the *openLimitReached* could not be 1 and both sensors could not be 1 together. If any of these erroneous situations occurs a transition is triggered to an error state. This behaviour is detailed in the parallel region called *SensorErrorMonitoring* of the state machine.

For detecting errors during opening or closing, in parallel with the normal functioning, the time that remains in *opening* or *closing* states is monitored and if this time is bigger than a specified timeout a transition is triggered to an error state. This behaviour is detailed in the parallel region called *OpeningClosingTimeMonitoring* of the state machine.

As critical operations, we focus on the control of doors. In the following, we present the UML Sequence Diagrams (UML-SD) for the opening and closing of doors.

Figure 7 depicts the UML-SD for door opening scenario (normal behaviour). When a train driver requests the opening of doors, the TCMS first checks whether the train status is suitable for opening the doors without risk, i.e., the train is really stopped. Whether this safety constraint is fulfilled, the “enable door” command is sent to the `DoorController` component. Then, the `DoorController` component opens the door when enabled and upon passenger’s request, which is sent when a passenger press the interior/exterior opening door button.

Similarly, the door closing scenario is shown in Figure 8. When the driver commands doors closing, the TCMS system sends the “not enable door” command to the `DoorController` component. The `DoorController` component disables the door and closes it when the operation can be safely completed, i.e., there is no any obstacle detected. Otherwise,

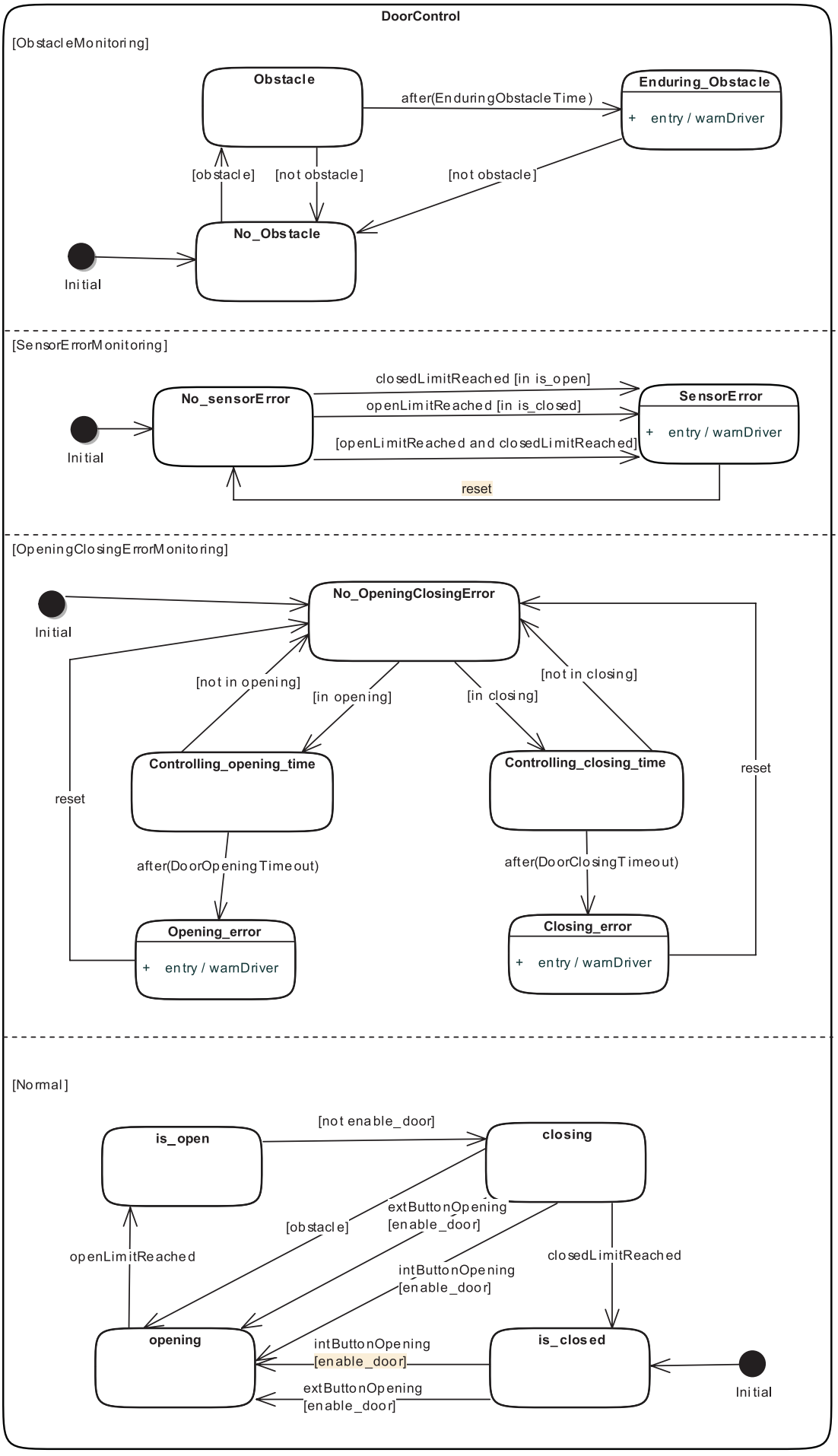


Fig. 6 UML State Machine Diagram of the DoorControl

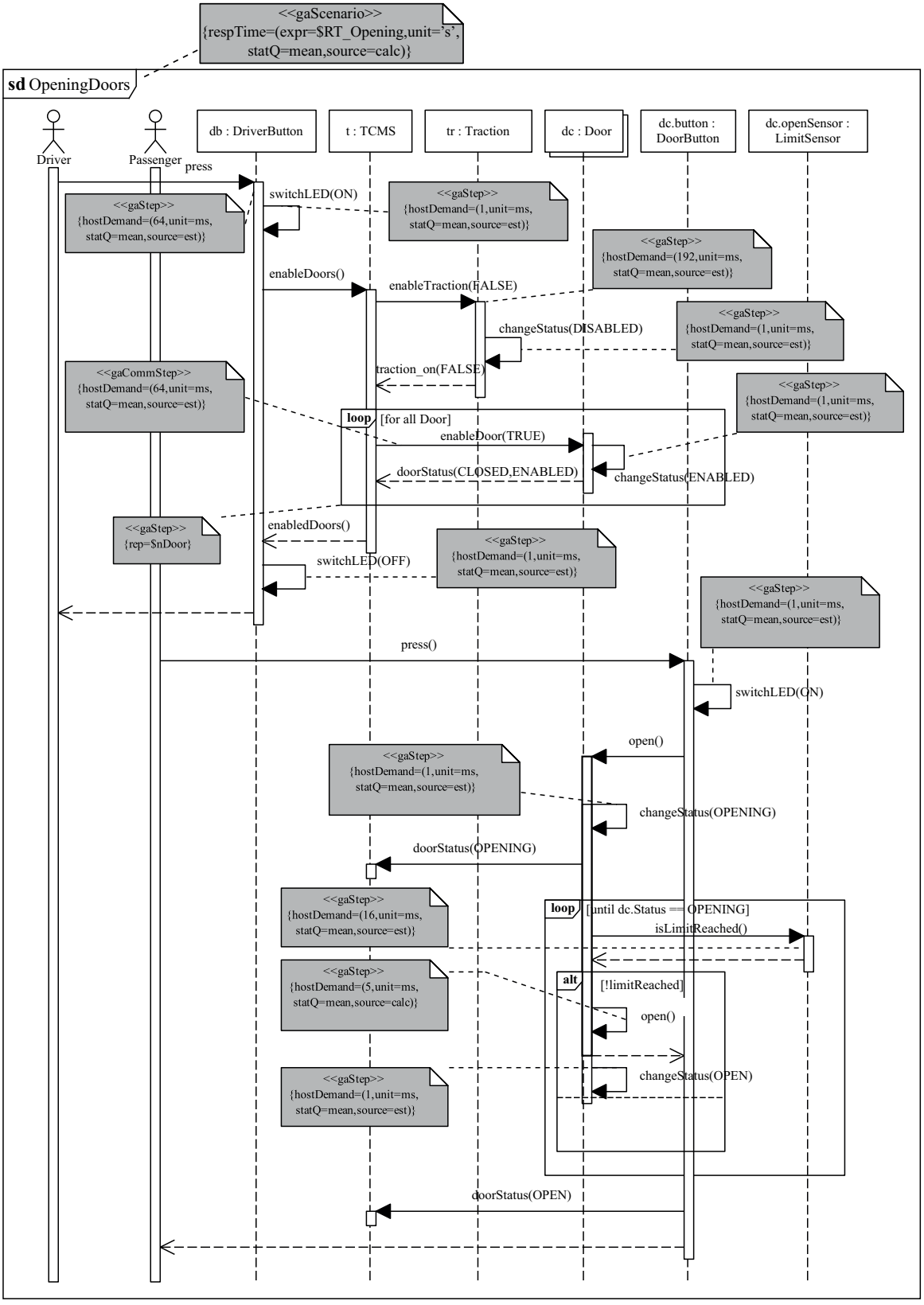


Fig. 7. UML Sequence Diagram representing the door opening operation.

the door is opened, and closing operation is again carried out. Recall that this closing/opening loop occurs until the door can be safely closed.

4.3. On Specification Phase

On Performance Specification Phase. Concerning our case study, safety engineers estimate the following performance objectives (PO) that the system must meet:

PO1. Response time of the Opening Doors scenario should be between 3 and 5 seconds.

PO2. Response time of the Closing Doors scenario should be between 3 and 5 seconds.

These performance requirements are collected by two predicted variables named $\$RT\_Opening$ and $\$RT\_Closing$, respectively, in the `gaScenario` stereotype of MARTE (see upper grey-highlighted note in Figures 7 and 8). In this case, response time is a measure to be calculated during analysis as indicated by *source=calc*. The `unit` of measurement are seconds and the *statistical measure* is a mean. Performance information use as duration activities within the UML model are gathered from technical specifications of real industrial train systems.

On Safety Specification Phase. A safety engineer defines the following safety requirements (SR) in the context of this case study:

SR1. When a door is enabled, traction is off, and a passenger press the opening button, the door starts to open unless the door is already open.

SR2. When an obstacle is detected and the door is closing, it starts to open.

SR3. When the door is enabled and the close event is received, the door starts to close unless the door is already closed. The close event is translated by the `TCMS` and sent to the `Door` component as $\neg$ *enable_door*.

In this phase, these requirements are expressed in terms of SCFs considering the component-based system depicted in Figure 5:

- $\mathcal{SR}_1 = \langle door\_enabled \wedge \neg traction\_on \wedge (inButtonOpening \vee extButtonOpening), doorStatus = OPENING \rangle$, defined on the `TCMS` component.
- $\mathcal{SR}_2 = \langle obstacle, doorStatus = OPENING \vee doorStatus = IS\_OPEN \rangle$. In this case, defined on the `DoorController` component.
- $\mathcal{SR}_3 = \langle door\_enabled \wedge \neg enable\_door, doorStatus = CLOSING \rangle$. This SCF is defined also on the `TCMS` component.

Since we do not distinguish between internal or external button opening in our Petri net model, in the sequel we consider $inButtonOpening \vee extButtonOpening$ as a single event named $openButton$. Note that $\mathcal{SR}_1, \mathcal{SR}_3$, requirements are defined on the `TCMS` component, while the context of $\mathcal{SR}_2$ is the `DoorController` component since the input and output ports that relate the $\mathcal{SR}_2$ belong to `DoorController`. As we have previously defined in Section 3.2, assumptions and guarantees of an SCF relate input and output ports of the components where they are defined.

Following Definition 4, these SCF are transformed into OCL, embedded within the UML-CD of the system. Namely, the constraints expressed as OCL language are shown at Code 1. Each OCL rule in that listing corresponds respectively to each one of the aforementioned SCF.

The context of OCL is directly taken from where SCF are defined. Finally, these OCL rules are transformed into Petri nets and integrated within the GSPN of the case study, as we explain in the next section.

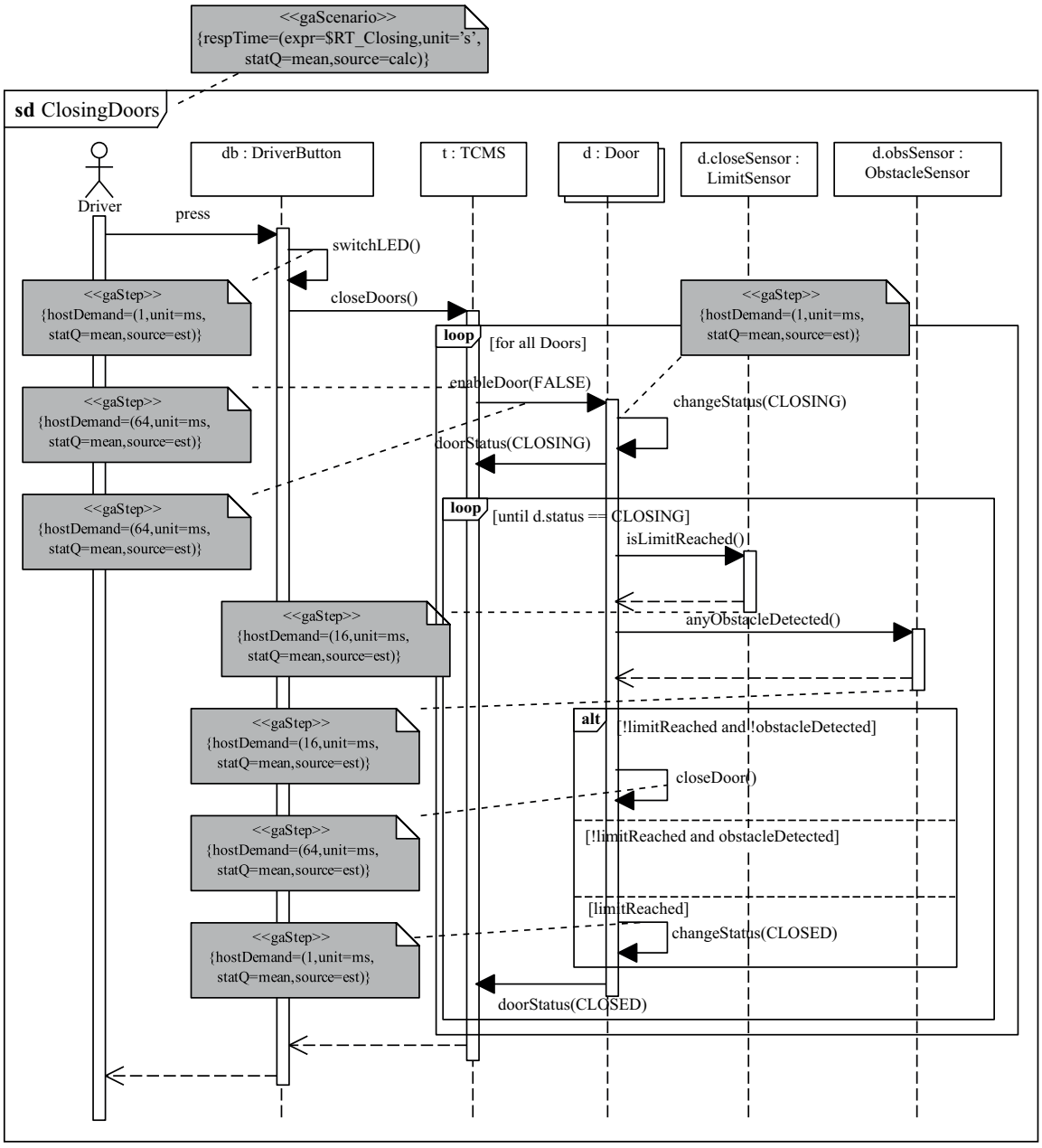


Fig. 8. UML Sequence Diagram representing the door closing operation.

Code 1. OCL constraints obtained from SCF transformation.

```
context TCMS
  inv: door_enabled and  not traction_on and open_button
                implies doorStatus = OPENING

context DoorController
  inv: obstacle
                implies (doorStatus = OPENING or doorStatus = IS_OPEN)

context TCMS
  inv: door_enabled and not enable_door
                implies doorStatus = CLOSING
```

4.4. On Transformation Phase

Following the methodology proposed in Section 3, this phase encompasses the transformation of the performance and safety scenarios described by UML-SDs, enriched with MARTE and OCL constraints, into GSPNs.

For this purpose, we use the ArgoSPE tool developed by [38]. Figure 9 depicts the GSPN obtained after transformation of UML-SD shown in Figures 7 and 8. The left-hand side of the figure represents the door opening scenario, while the right-hand side represents the door closing. The transformation process is partially done in an automatic way by ArgoSPE, since OCL constraints transformation is unsupported by the current release of ArgoSPE and thus some manual tuning is needed.

Some additional modifications are made manually. Specifically, those related to other elements of the system that are not completely considered in the first Safety-Oriented Design Phase. In particular, we have also modelled the Traction operation without considering human interaction, thus, our system automatically speeds up after closing the door and it brakes when the traction receives a traction stop signal.

OCL constraints described in the previous section are now transformed into Petri nets, following the guidelines given in Section 3.3. The Petri nets generated from $\mathcal{SR}_1$, $\mathcal{SR}_2$, and $\mathcal{SR}_3$ are depicted in Figure 10. Let us briefly exemplify how a PN representing an OCL constraint is built. Recall that we represent the violation of the safety condition as a Petri net. Consider OCL constraint $TCMS\_SR1$. Applying our transformation, such an invariant is violated when $(door\_enabled \wedge \neg traction\_on \wedge open\_button \wedge \neg(doorStatus = OPENING)$ if fulfilled. A place is generated to represent each of the clauses, and extra places/transitions are added to join them into a place that represents the OCL constraint (place $\neg\mathcal{SR}_1$, in this case, see Section 3.3).

Our aim during analysis is to check whether the places $\neg\mathcal{SR}_1$, $\neg\mathcal{SR}_2$, and $\neg\mathcal{SR}_3$ are marked in a single run execution of the net, thus indicating that safety conditions are not fulfilled. Recall that the probability of (eventually) violating a safety condition (i.e., an invariant) is represented as a place being (eventually) marked.

These nets can finally be merged with the PN of the safety scenarios depicted in Figure 9. Both nets are merged using the transitions that create tokens in places representing the same issue, i.e., places $tractionOn$ and $doorStatusOPENING$ in Figure 10 represent the same state than $p\_traction\_on\_TRUE$ and $p\_door\_CLOSING$, respectively, in Figure 9. The connection to places representing safety contracts have been highlighted (grey colour) in Figure 9. Recall that places in Figure 10 depicted with a dotted line represent the negation of an assumption or guarantee and thus we should check whether these places are marking or not.

4.5. On Analysis Phase

In this step, the software engineer reviews the performance and safety objectives, which were defined during the design step, and carries out the analysis of the performance and safety model.

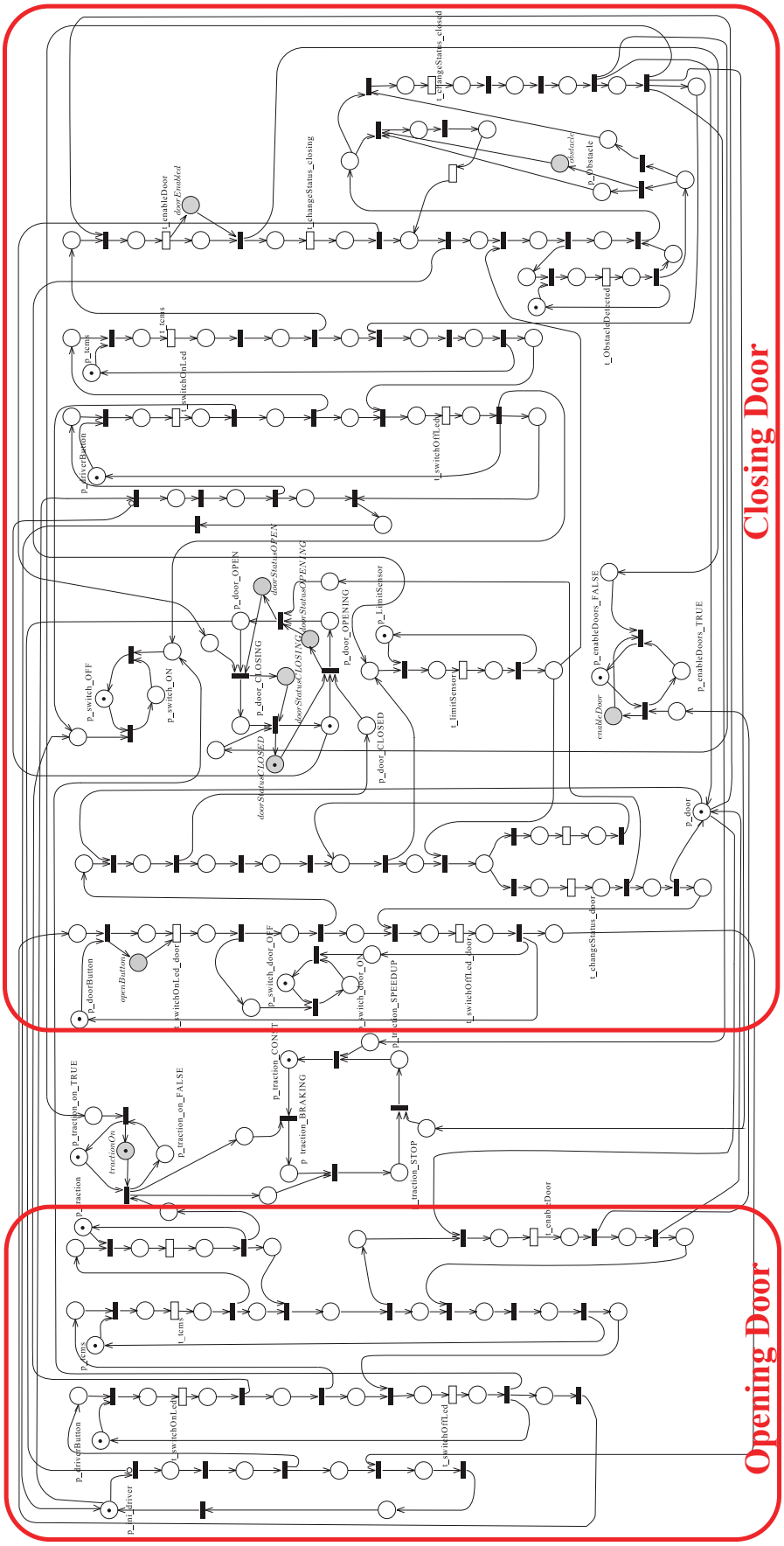
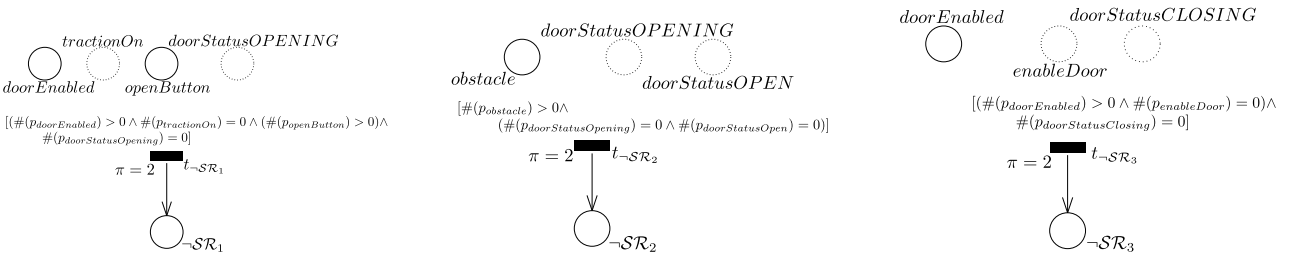


Fig. 9. Petri net corresponding to the opening and closing of a door.



(a) OCL constraint `TCMS_SR1` (b) OCL constraint `DoorController_SR2` (c) OCL constraint `TCMS_SR3`

Fig. 10. Petri net representation of OCL constraints of the case study.

Regarding performance requirements, as determined in Section 4.3, the system has two performance objectives `PO1` and `PO2`: response times between 3 and 5 seconds for both scenarios, i.e., Opening and Closing Doors. Concerning safety requirements, the system must fulfil safety objectives `TCMS_SR1`, `DoorController_SR2` and `TCMS_SR3` described by means of OCL constraint in Section 4.3.

Therefore, the merged PN depicted in Figure 9 is finally analysed to obtain measures of interest from performance and safety viewpoint. We use the GreatSPN tool [25] to compute these objectives. The response time of a scenario is calculated as the inverse of the throughput of the transition that closes the entire execution cycle (see transitions *end_cycle* in Figure 9). The computed response times are 3.4 seconds and 4.2 seconds for Opening Doors and Closing Doors scenarios, respectively.

On the other hand, we also compute the probability of places $\neg\mathcal{SR}_1, \neg\mathcal{SR}_2, \neg\mathcal{SR}_3$ having a marking greater than zero. When this situation occurs, it indicates that the OCL constraints `TCMS_SR1`, `DoorController_SR2`, and `TCMS_SR3` are not fulfilled. A single run execution of the net is performed, and returns zero values for these probabilities, thus safety contracts are fulfilled in the system model. Let us finally remark that final effort must be focused on assuring that the system implementation matches the UML models. Otherwise, although a safety verification of models have been proved, the system may reach unsafe states.

Note that the UML models that we described here are enriched with MARTE profile annotations to carry out performance analysis, but these enriched data are not used for the safety analysis. However, these data can be necessary for verifying some safety properties where timing become relevant [50]. To this aim, we may use OCL/RT [51], an extension of native OCL to specify time issues, in conjunction with the MARTE profile, and translate such an information into the GSPN models. We consider this an interesting issue which deserves further study.

4.6. On Assessment Phase

According to our methodology detailed in Sect. 3, this phase proposes alternatives for getting an "optimal system configuration" for the Train Doors Controller system. These alternatives include resource replication (using utilization resource analysis) and application of performance patterns and antipatterns, in the performance perspective. As commented in Sect. 3, we are analysing how to systematize alternatives to improve the system from the safety point of view.

Since in the case study both performance objectives and safety requirements are met, the assessment phase does not yield further improvements. Therefore, the initial configuration is the "optimal configuration".

5. Discussion

The assessment of software system is a process that is acquiring increasing importance in industrial practice. The work carried out allowed us to determine a design and configuration that meet safety and performance requirements and, therefore, to improve the final product. In the following, we discuss limitations of the outcomes obtained, lessons learned and issues disclosed while applying the methodology, we also explain some of the consequences of all these matters.

We can interpret the results from two perspectives. On the one hand, we discuss the outcomes explicitly related to the methodology. On the other hand, we analyse the collected data obtained by applying the methodology to the proposed case study in order to achieve an optimal configuration.

5.1. Concerning the Methodology

The ultimate aim of our research is to achieve a unique process to evaluate software systems in order to meet non-functional requirements. Following this rationale, the proposed methodology extends the performance assessment proposed in [20] to include other feature, in this case, specifically safety requirements. The proposed methodology is therefore intended to support safety-critical domains that meet both performance and safety requirements.

Concerning the process, the methodology explicitly influences the development process by focusing on safety and performance properties. The methodology systematically defines all the steps needed to discover potential safety and performance problems and how to mitigate them.

Bass et al. [52] determine that quality attributes can never be achieved in isolation, the achievement of any one will have an effect, sometimes positive and sometimes negative, on the achievement of others. As pointed in [20], when we applied our methodology, the improvement of system safety and performance could influence other quality attributes, such as maintainability or cost, and in the worst case, the improvement of safety and performance could decrease other quality attributes. Specifically, since the benefit of using design patterns for improving the quality of a system is widely recognized, the use of patterns and anti-patterns during the assessment phase influence its maintainability.

Concerning the modelling criterion, it analyses how the performance and safety requirements and system functionalities are specified and developed. This approach is focused on the components level and captures the system structure model and its behaviour, then allowing its evaluation from the performance and safety point of view.

Concerning the tools, we used ArgoSPE [38], an ArgoUML[3] plugin, to partly automate the process in a transparent way for software engineers. Although the functionality of this tools is very limited, it is very useful since it allows us to automatically translate some UM diagrams into GSPNs, specifically UML-CD, UML-SD and UML-SM. Furthermore, it does not support performance annotations in MARTE, but in a UML previous profile format Thereby, we had the choice of translating into these annotations or to introduce some performance parameters in the GSPN manually, as observed in [20]. ArgoSPE lacks other plugins, such as tools for identifying performance patterns and anti-patterns automatically. Regarding safety perspective, the translation of OCL contracts into GSPNs is carried out manually, since this functionality is not implemented yet. We have detected the need for developing a new framework which integrates all these functionalities: UML modelling, MARTE annotations, OCL contracts, patterns and anti-patterns detection and GSPN simulation and analysis in a transparent way to the user and efficient computation times. This framework could also include assessment of other functional and non-functional properties, such as dependability, security or model checking.

5.2. Concerning the Case Study

The methodology has been applied to an industrial case study in railway domain, described in Sect. 4. The real case study is a complex distributed system that controls many train subsystems, called TCMS. With the application of our methodology, we can verify that this TCMS meets performance and safety requirements in the early phase of design.

However, some limitations are still unsolved. First of all, in real implementations, the interaction between subsystems is usually complex. In our case, for the sake of clarity and comprehension, we have just focused on door controlling. Some simplifications have been done: interaction with other components of the TCMS, dependencies with other subcomponents, and their communication were omitted. Another important aspect in real implementations is fault-tolerance and how the

[3] http://argouml.tigris.org/

transition to a fail-safe state of the system is made. In our case, the main focus has been in normal behaviour. Possible environmental faults have been modelled but without describing the transition to a fail-safe state of the system. Similarly, software and hardware fault tolerance techniques has not been considered. In this regard, Petri nets patterns introduced in [53] may help to improve our approach.

6. Related Work

Several methodologies have been proposed for the verification of safety properties on critical systems, in particular, the following propose similar methodologies to the one we propose, see [54; 55]. In [54], contract-based design is used for "static and dynamic verification of components' compatibility". As in our case, contracts are defined within UML models by using OCL but, in contrast to our approach, there is neither a formal definition of contracts, nor a formal translation of contracts into OCL. Verification in [54] is achieved by means of flow graphs, an intermediate language between DMOSES models and the input languages of model checkers. In contrast, in our approach, UML models enriched with MARTE annotations and OCL constrains are directly translated to GSPN. The advantage in this case is that there is only one translation step.

The AVATAR methodology presented in [55] comprises similar stages to the one we propose, namely: requirement capture, system analysis, system design, property modelling, and formal verification by means of model checking. A key difference with our work is that properties are expressed in the TEPE (Temporal Property Expression Language) created by the authors. TEPE seems very well suited for the verification of real-time critical systems since it enriches the expressiveness of other typical property languages with the notion of physical time and unordered signal reception. However, an advantage of our approach is that we use OCL for safety specification and MARTE for performance specification, both well-known languages in the software engineering community.

Our methodology extends the performance analysis methodology presented in [20], which is based on principles and techniques of Software Performance Engineering (SPE) [23]. Concerning methodologies based on SPE principles, to the best of our knowledge, there are very few initiatives and all of them are focussed on performance analysis. For instance, the PASA (Performance Assessment of Software Architectures) method, proposed by [56] is a performance scenario-based software architecture analysis method that provides a framework for the whole assessment process. Nevertheless, some steps of PASA entrust in the software engineer expertise to be applied and to identify alternatives for improvements. [57] defined Continuous Performance Assessment of Software Architecture (CPASA). This method adapts PASA to the agile development process. Unlike our proposal, these methodologies only analyse performance issues.

Regarding contracts, many formalisms have been proposed to express contracts, such as the Requirements Specification Language (RSL) [5], the Othello language [6], which is based on Linear Temporal Logic, or Modal Transmission Systems [58]. Unlike OCL, these languages are more expressive but OCL is a well-known language among modelisation engineering community. However, a major drawback of these formalisms is that the requirement engineers need to learn a new formalism each time they need to write contracts in a specific domain. In contrast, OCL is a well-known language in industry. Besides, to the best of our knowledge some of the proposed formalisms lack the means to verify that a component model fulfils their contracts [5; 58], or only focus on verification of functional properties [6]. In this work, we have shown that OCL contracts can be used to perform safety assessment by translating the UML models to Petri nets. Although currently we also focus on functional properties, the use of UML profiles enables to analyse other non-functional properties that can affect to safety, such as performance, dependability or security.

Representing safety contracts using OCL has been previously proposed in [50]. The novelty of our work is that we propose a translation from safety contracts in the form of assumptions and guarantees to OCL. Our work complements the work of OTHELLO language [6] and OCRA [59]. In particular, the analysis of non-functional properties can complement the work on verifying functional properties in OCRA [59]. Other work similar to ours is [60], where UML/OCL is used to

express system invariants, transformed into Place/Transition nets (without time) and to LTL logic for the verification. In contrast to their work, we formalise the safety contracts, and, moreover, our Petri net models capture the timing information.

Some works refine safety contract assumptions in strong and weak assumptions [5; 61]. Strong assumptions specify what always is fulfilled by the environment, context-independently, while weak assumptions provide additional information about the context where a component could operate (e.g., the expected timing between input signals). In this paper, we consider the definition of safety contract as given in [24], having only strong assumptions. In our case, the weak assumptions can be implicitly described by UML annotations. As future work, we aim at extending our safety contract specification to explicitly express timing issues.

7. Conclusions and Future Work

Safety-critical systems need to assure that safety requirements are fulfilled before they are deployed. Safety assessment normally relies on methods applied during the normal operation of the system, detecting design problems that lead to a non-compliance with the system safety requirements. Thus, these systems are redesigned, normally inducing a huge cost overhead since the development process has to be is redone. To alleviate this problem, safety assessment should be performed in early stages of the development lifecycle.

To this aim, in this paper we present a model-based methodology for the safety assessment of critical systems. The methodology consists of three phases: Safety-oriented design, Safety-oriented specification, and Safety-oriented analysis. In the safety-oriented design phase, a component-based UML model is sketched using Composite Structure Diagrams and Sequence Diagrams. These diagrams capture the structure and behaviour of the software systems to be analysed. In the safety-oriented specification phase, safety requirements are first expressed as Safety Contracts Fragments, and then translated into OCL. Finally, in the safety-oriented analysis phase, the aforementioned UML models enriched with OCL constraints obtained from the previous phase are translated into a formal model (in particular, Generalized Stochastic Petri nets), where the analysis is carried out. The proposed methodology is evaluated with a real industrial case study from the railway domain, namely, a train door controller system.

Our methodology is an extension of the methodology for performance analysis presented in [20]. The advantage of extending this existing methodology is clear, since performance and safety can be assessed using a similar approach.

The specification of safety contracts in terms of OCL within UML models allows to express safety requirements and system characteristics in a single picture. The adoption of formal models in early stages of development lifecycle, obtained after the transformation of UML/OCL models into Petri nets, ensures an early verification of fulfilment of safety requirements. Besides, this early adoption becomes also easy since UML/OCL are modelling languages familiar to the industry engineers community. This approach complements other approaches where more expressive languages than OCL, for instance, Linear Temporal Logic (LTL), are used to specify safety contracts. In our approach, we sacrifice expressiveness in favour of simplicity in the contracts specification language. However, this issue can be overcome by extending OCL with more operators, for instance, temporal operators. Similarly, the lack of specifying event sequences and checking error propagation can also be overcome by extending OCL semantics. We aim at extending OCL covering these issues as future work.

Another interesting aspect that deserves further study is the analysis of safety contracts where concrete time values are considered, for instance, “a door is closed within 5 seconds when the door opening is enabled and the close event is received”. To allow the analysis of such time constraints, we would need to extend the safety analysis phase of our proposed methodology. Specifically, we would need to provide the translation process from OCL constraints into Petri nets with time intervals.

Acknowledgements

The authors are grateful to anonymous referees for providing constructive comments and helping to improve the contents of this manuscript. The research of the authors was supported in part by the ARTEMIS Joint Undertaking under grant agreement no. 295373 (project nSafeCer) and by National funding. The research of Ricardo J. Rodríguez was also supported in part by EU Horizon 2020 research and innovation programme under grant agreement no. 644869 (DICE) and by Spanish MINECO project CyCriSec (TIN2014-58457-R). The research of Clara Benac Earle was also supported by Spanish MINECO project STRONGSOFT (TIN2012-39391-C04-02) and by the Madrid Regional Government project nGreens (S2013/ICE-2731).

References

[1] nSafeCer project. Safety Certification of Software-Intensive Systems with Reusable Components. Project Grant Agreement n^o 295373. http://safecer.eu/, (2015, accessed 15 February 2015).

[2] Lutz RR. Software Engineering for Safety: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering*. ICSE '00, New York, NY, USA: ACM. ISBN 1-58113-253-0, pp. 213–226. .

[3] Grottke M and Trivedi K. Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate. *Computer* 2007; 40(2): 107–109. .

[4] Feiler PH. Model-Based Validation of Safety-Critical Embedded Systems. In *IEEE Aerospace Conference (AERO)*. pp. 1–10. .

[5] Damm W, Hungar H, Josko B et al. Using Contract-based Component Specifications for Virtual Integration Testing and Architecture Design. In *Proceedings of the Design, Automation Test in Europe Conference Exhibition (DATE)*. pp. 1–6. .

[6] Cimatti A and Tonetta S. A Property-Based Proof System for Contract-Based Design. In *Proceedings of the 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. pp. 21–28. .

[7] Kath O, Schreiner R and Favaro J. Safety, Security, and Software Reuse: A Model-Based Approach. In *Proceedings of the Fourth International Workshop in Software Reuse and Safety*.

[8] OMG. Unified Modeling Language (UML), 2011. Version 2.4.1, http://www.omg.org/spec/UML/2.4.1/, accessed 10 February 2015.

[9] OMG. *Object Constraint Language (OCL)*. Object Management Group, 2010. V2.2, formal/2010-02-01.

[10] OMG. UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms (QoS & FT), 2008. Version 1.1, http://www.omg.org/spec/QFTP/, accessed 9 February 2015.

[11] Rodríguez RJ and Gómez-Martínez E. Model-based Safety Assessment using OCL and Petri Nets. In *Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. pp. 56 – 59. .

[12] Murata T. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE* 1989; 77(4): 541–580.

[13] Gómez-Martínez E, Rodríguez RJ, Etxeberria L et al. Model-Based Verification of Safety Contracts. In Canal C and Idani A (eds.) *Software Engineering and Formal Methods - SEFM 2014 Collocated Workshops: HOFM, SAFOME, OpenCert, MoKMaSD, WS-FMDS, Revised Selected Papers*, *Lecture Notes in Computer Science*, volume 8938. Springer, pp. 101–115. .

[14] International Organization for Standardization. ISO/IEC 19505-1: Information technology – Object Management Group Unified Modeling Language (OMG UML) – Part 1: Infrastructure, 2012.

[15] OMG. A UML profile for Modeling and Analysis of Real Time Embedded Systems (MARTE), 2011, http://www.omgmarte.org/, accessed 4 February 2015. Version 1.1.

[16] Bernardi S, Merseguer J and Petriu DC. Dependability modeling and analysis of software systems specified with UML. *ACM Comput Surv* 2012; 45(1): 2.

[17] Rodríguez RJ, Merseguer J and Bernardi S. Modelling and Analysing Resilience As a Security Issue Within UML. In *Proceedings of the 2nd International Workshop on Software Engineering for Resilient Systems*. SERENE '10, New York, NY, USA: ACM. ISBN 978-1-4503-0289-0, pp. 42–51. .

[18] Ajmone Marsan M, Balbo G, Conte G et al. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing, John Wiley and Sons, 1995.

[19] Bernardi S and Merseguer J. Performance evaluation of UML design with Stochastic Well-formed Nets. *Journal of Systems and Software*

2007; 80(11): 1843–1865.

[20] Gómez-Martínez E, González-Cabero R and Merseguer J. Performance assessment of an architecture with adaptative interfaces for people with special needs. *Empirical Software Engineering* 2014; 19(6): 1967–2018. .

[21] Gómez-Martínez E, Ilarri S and Merseguer J. Performance Analysis of Mobile Agents Tracking. In *Proc. 6th Int. Workshop on Software and Performance (WOSP'07)*. ACM. ISBN 1-59593-297-6, pp. 181–188. .

[22] Gómez-Martínez E and Merseguer J. Performance Modeling and Analysis of the Universal Control Hub. In *Proc. the 7th European Performance Engineering Workshop (EPEW'10)*, *Lecture Notes in Computer Science*, volume 6342. Springer, pp. 160–174.

[23] Smith CU. Increasing Information Systems Productivity by Software Performance Engineering. In *Proc. 7th Int. Conf. Computer Measurement Group (CMG'81)*. pp. 5–14.

[24] Söderberg A and Johansson R. Safety Contract Based Design of Software Components. In *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. pp. 365–370. .

[25] Baarir S, Beccuti M, Cerotti D et al. The GreatSPN tool: recent enhancements. *SIGMETRICS Perform Eval Rev* 2009; 36(4): 4–9.

[26] Sljivo I, Gallina B, Carlson J et al. A Method to Generate Reusable Safety Case Fragments from Compositional Safety Analysis. In Schaefer I and Stamelos I (eds.) *Software Reuse for Dynamic Systems in the Cloud and Beyond - 14th International Conference on Software Reuse, ICSR 2015, Miami, USA, January 4-6, 2015*, *Lecture Notes in Computer Science*, volume 8919. Springer, pp. 253–268. .

[27] OMG. Systems Modeling Language (SysML), 2012. Version 1.3, http://www.omgsysml.org/, accessed 13 February 2015.

[28] Sangiovanni-Vincentelli A, Damm W and Passerone R. Taming Dr. Frankenstein: Contract-based design for cyber-physical systems. *European Journal of Control* 2012; 18(3): 217–238.

[29] Gómez-Martínez E. *Software Performance Assessment at. Architectural Level: a Methodology and its Application*. Phd thesis, Universidad de Zaragoza, 2014. Http://zaguan.unizar.es/record/13510/files/TESIS-2014-021.pdf.

[30] Lazowska ED, Zahorjan J, Graham GS et al. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Prentice-Hall, 1984.

[31] Hermanns H, Herzog U and Katoen JP. Process Algebra for Performance Evaluation. *Theoretical Computer Science* 2002; 274(1-2): 43–87. .

[32] Petriu DC and Woodside M. Software Performance Models from System Scenarios in Use Case Maps. In *Proc. 12th Int. Conf. on Computer Performance Evaluation, Modelling Techniques and Tools (TOOLS'02)*, *Lecture Notes in Computer Science*, volume 2324. Springer, pp. 141–158.

[33] Tribastone M and Gilmore S. Automatic Translation of UML Sequence Diagrams into PEPA Models. In *Proc. 5th Int. Conf. on the Quantitative Evaluation of Systems (QEST'08)*. IEEE Computer Society, pp. 205–214.

[34] Bernardi S and Merseguer J. Performance evaluation of UML design with Stochastic Well-formed Nets. *J Syst Softw* 2007; 80(11): 1843–1865. .

[35] Distefano S, Scarpa M and Puliafito A. From UML to Petri Nets: The PCM-Based Methodology. *IEEE Trans Softw Eng* 2011; 37(1): 65–79.

[36] Zimmermann A. Modeling and evaluation of stochastic Petri nets with TimeNET 4.1. In *Proceedings of the 6th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS)*. pp. 54–63.

[37] Rodríguez RJ, Júlvez J and Merseguer J. PeabraiN: A PIPE Extension for Performance Estimation and Resource Optimisation. In *Proceedings of the 12th International Conference on Application of Concurrency to System Designs (ACSD)*. IEEE, pp. 142–147. .

[38] Gómez-Martínez E and Merseguer J. ArgoSPE: Model-based Software Performance Engineering. In *Proc. 27th Int. Conf. on Applications and Theory of Petri Nets and Other Models of Concurrency (ICATPN'06)*, *Lecture Notes in Computer Science*, volume 4024. Springer, pp. 401–410. . Tool available at: http://argospe.tigris.org.

[39] Delatour J and de Lamotte F. ArgoPN: a CASE Tool Merging UML and Petri Nets. In *Proc. 3rd Int. Workshop on New Developments in Digital Libraries and the 1st Int. Workshop on Validation and Verification of Software for Enterprise Information Systems (NDDL/VVEIS'03)*. ICEIS Press. ISBN 972-98816-2-6, pp. 94–102.

[40] López-Grao JP, Merseguer J and Campos J. From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering. In *Proc. 4th Int. Workshop on Software and Performance (WOSP'04)*. ACM, pp. 25–36.

[41] Smith CU. *Performance Engineering of Software Systems*. Addison–Wesley, 1990.

[42] Smith CU and Williams LG. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison–Wesley, 2002.

[43] Smith CU and Williams LG. Software Performance Antipatterns. In *Proc. 2nd Int. Workshop on Software and Performance (WOSP'00)*. ACM, pp. 127–136.

[44] CENELEC. EN50126 Railway applications - The specification and demonstration of Reliability Availability Maintainability and Safety (RAMS), 1999.

[45] CENELEC. EN50128 Railway applications - Communications signalling and processing systems - Software for railway control and protection systems, 2001.

[46] CENELEC. EN50129 Railway applications - Communication signaling and processing systems - Safety related electronic systems for signaling, 2003.

[47] International Electrotechnical Comission. IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems, 2010.

[48] Defence standard 00-56 issue 4: Safety management requirements for defence systems, 2007.

[49] Safecer. Deliverable d2.1.4 & d4.1.2: Standards-oriented, domain-specific aspects in reusing software in safecer domains. http://www.safecer.eu/images/pdf/pSafeCer_Deliverable_D4_1_2.pdf, 2013.

[50] Bate I, Hawkins R and McDermid J. A Contract-based Approach to Designing Safe Systems. In *Proceedings of the 8th Australian Workshop on Safety Critical Systems and Software - Volume 33*. SCS '03, Australian Computer Society, Inc. ISBN 1-920-68215-5, pp. 25–36.

[51] Cengarle MV and Knapp A. Towards OCL/RT. In *FME 2002: Formal Methods – Getting IT Right*, *Lecture Notes in Computer Science*, volume 2391. Springer Berlin Heidelberg. ISBN 978-3-540-43928-8, 2002. pp. 390–409. .

[52] Bass L, Clements P and Kazman R. *Software Architecture in Practice*. SEI Series in Software Engineering, Addison-Wesley, 2005. ISBN 0-321-15495-9.

[53] Rodríguez RJ, Júlvez J and Merseguer J. Quantification and Compensation of the Impact of Faults in System Throughput. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability* 2013; 227(6): 614–628. .

[54] Pérez ZAD. *Model-driven development methodology for hybrid embedded systems based on UML with emphasis on safety-related requirements*. Phd, Faculty of Electrical Engineering and Computer Science, University of Kassel, 2014.

[55] Knorreck D, Apvrille L and de Saqui-Sannes P. Tepe: A sysml language for time-constrained property modeling and formal verification. *SIGSOFT Softw Eng Notes* 2011; 36(1): 1–8. .

[56] Williams LG and Smith CU. PASA^{SM}: A Method for the Performance Assessment of Software Architectures. In *Proc. 3rd Int. Workshop on Software and Performance (WOSP'02)*. ACM, pp. 179–188.

[57] Pooley RJ and Abdullatif AAL. CPASA: Continuous Performance Assessment of Software Architecture. In *Proc. 17th IEEE Int. Conf. and Workshops on the Eng of Computer-Based Systems (ECBS'10)*. IEEE Computer Society. ISBN 978-0-7695-4005-4, pp. 79–87.

[58] Bauer SS, David A, Hennicker R et al. Moving from Specifications to Contracts in Component-Based Design. In Lara J and Zisman A (eds.) *Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering (FASE)*, *Lecture Notes in Computer Science*, volume 7212. Springer Berlin Heidelberg. ISBN 978-3-642-28871-5, 2012. pp. 43–58. .

[59] Cimatti A, Dorigatti M and Tonetta S. OCRA: A tool for checking the refinement of temporal contracts. In *28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, pp. 702–705.

[60] Bouabana-Tebibel T and Belmesk M. Integration of the Association Ends within UML State Diagrams. *Int Arab J Inf Technol* 2008; 5(1): 7–15.

[61] Sljivo I, Gallina B, Carlson J et al. Strong and Weak Contract Formalism for Third-Party Component Reuse. In *IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*. pp. 359–364. .